Модел. и анализ информ. систем. Т. **20**, № **2** (2013) 5–22

УДК 519.172

Ассоциативный параллельный алгоритм для динамической обработки дерева кратчайших путей

Непомнящая А. Ш.

Институт вычислительной математики и математической геофизики СО РАН
630090, Россия, г. Новосибирск, проспект Академика Лаврентьева, 6

e-mail: anep@ssd.sscc.ru

получена 19 апреля 2012

Ключевые слова: ориентированный взвешенный граф, дерево кратчайших путей, матрица смежности, декрементальный алгоритм, ассоциативный параллельный процессор

В работе строится эффективный ассоциативный параллельный алгоритм для динамической обработки дерева кратчайших путей после удаления одной дуги из ориентированного взвешенного графа. С этой целью вначале описывается используемая структура данных и ее построение, а также приводится STAR–машина, которая моделирует работу ассоциативных (контекстно–адресуемых) параллельных систем с простыми процессорными элементами и вертикальной обработкой информации. На этой модели ассоциативный параллельный алгоритм представляется в виде главной процедуры DeleteArcSPT, использующей группу вспомогательных процедур для выполнения его отдельных частей. Доказана корректность процедуры DeleteArcSPT и показано, что на STAR-машине она выполняется за время $O(hk)$, где h – число битов, которое требуется для кодирования длины максимального кратчайшего пути, а k – число вершин, для которых вычисляются новые кратчайшие пути после удаления одной дуги из исходного графа.

1. Введение

Задача нахождения кратчайших путей возникает в различных приложениях. Известны две версии этой проблемы: нахождение кратчайших путей из одной вершины (the single-source shortest paths problem) и нахождение кратчайших путей между каждой парой вершин (the all-pairs shortest paths problem). Динамическая версия проблемы нахождения кратчайших путей из одной вершины состоит в обработке информации о кратчайших путях во время изменения графа без перевычисления графа целиком после каждого локального изменения в нем. Типичными операциями для этого вида преобразований являются добавление или удаление одной дуги либо изменение веса одной дуги. Если граф представляет коммуникационную или транспортную сеть, то добавление или удаление дуги отражает такие реальные изменения

в сети как добавление или удаление связей во время существования сети. Алгоритм называется *полностью динамическим* (fully dynamic), если он позволяет выполнять любую последовательность упомянутых выше операций. Алгоритм называется *инкрементальным*, если он допускает только добавление дуги или уменьшение веса дуги. Если алгоритм допускает только удаление дуги или увеличение веса дуги, то он называется *декрементальным*.

В работах [1, 2] для графов с произвольными вещественными весами дуг построены полностью динамические алгоритмы для обработки дерева кратчайших путей. При этом авторы полагают, что граф не имеет циклов отрицательной длины до и после обработки входной информации. В этих работах используется модель, которая учитывает сложность выходной информации после модификации входа. Сложность динамического алгоритма оценивается суммой числа аффектных вершин, которые модифицируют свое кратчайшее расстояние после преобразования входной информации, и числа дуг, у которых по крайней мере одна из вершин является аффектной. В работе [3] построены полудинамические алгоритмы для обработки кратчайших расстояний и дерева кратчайших путей как в ориентированном, так и в неориентированном графе с положительными вещественными весами ребер. Декрементальные алгоритмы строятся для планарных графов, а инкрементальные алгоритмы применяются к произвольным графам. Временная сложность алгоритмов оценивается в терминах амортизированного времени. В работе [4] построены полностью динамические алгоритмы для обработки кратчайших расстояний и дерева кратчайших путей как в ориентированном, так и в неориентированном графе с положительными действительными весами ребер после выполнения произвольной последовательности преобразований ребер. Стоимость операции обработки задается как функция от числа преобразований выходных данных, используя понятие k-ограниченной вычисляющей функции. В работе [5] строится полностью динамический алгоритм для обработки дерева кратчайших путей ориентированного графа с произвольными весами дуг. Авторы строят новый алгоритм для удаления дуги и для увеличения веса дуги и новый алгоритм для добавления дуги и для уменьшения веса дуги, в которых явно используются циклы отрицательной длины. Сложность операции обработки задается как функция от структуры графа и числа обработок выходной информации. В работе [6] построена эффективная параллельная реализация алгоритма Рамалингама [1] для динамической обработки подграфа кратчайших путей после удаления одной дуги из ориентированного взвешенного графа с выделенным стоком. В качестве модели вычисления используется STAR–машина [7], которая моделирует работу ассоциативных параллельных систем типа SIMD с последовательно–поразрядной (вертикальной) обработкой информации и простыми процессорными элементами. Исходное дерево кратчайших путей строится с помощью классического алгоритма Дейкстры [8]. Ассоциативная версия алгоритма Рамалингама представлена в виде процедуры DeleteArc, корректность которой доказана. Мы показали, что на STAR–машине эта процедура выполняется за время, которое пропорционально числу вершин, для которых изменяются кратчайшие расстояния после удаления одной дуги из графа. Следуя Фостеру [9], мы полагаем, что каждая элементарная операция STAR–машины выполняется за единицу времени. Также в работе [6] приведены основные достоинства параллельной реализации декрементального алгоритма Рамалингама.

В данной работе строится ассоциативный параллельный алгоритм для динамической обработки дерева кратчайших путей ориентированного графа с неотрицательными весами дуг после удаления из графа одной дуги. Ассоциативный параллельный алгоритм использует простую структуру данных, позволяющую выполнять доступ к данным по содержимому памяти. Алгоритм представлен в виде процедуры DeleteArcSPT, корректность которой доказана. Мы показали, что эта процедура выполняется на STAR–машине за время $O(hk)$, где h – число битов, которое требуется для кодирования максимума кратчайших расстояний от корня, а k – число аффектных вершин, для которых строятся новые кратчайшие пути и вычисляются новые кратчайшие расстояния. Отметим, что выполнение процедуры DeleteArcSPT проще, чем выполнение процедуры DeleteArc.

2. Модель ассоциативного параллельного процессора

Приведем краткое описание используемой модели. Подробное описание модели и сравнение ее с другими моделями ассоциативной обработки приводится в работе [10].

Модель определяется как абстрактная STAR-машина типа SIMD с вертикальной обработкой информации (Рис. 1), основными компонентами которой являются:

- последовательное устройство управления, в котором записаны программа и скалярные константы;
- устройство ассоциативной обработки, состоящее из p одноразрядных процессорных элементов;
- матричная память.

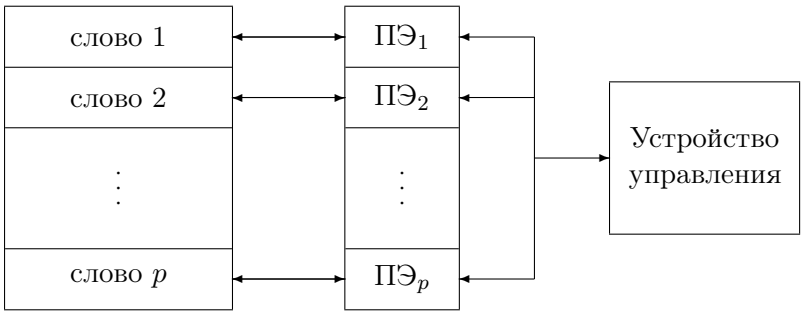

Рис. 1. STAR–машина

Входные данные, записанные в двоичном коде, помещаются в матричную память в виде двумерных таблиц, причем каждая единица данных хранится в отдельной строке и обрабатывается отдельным процессорным элементом (Рис. 2). Строки каждой таблицы нумеруются сверху вниз, а столбцы – слева направо. В матричную память можно загружать несколько таблиц.

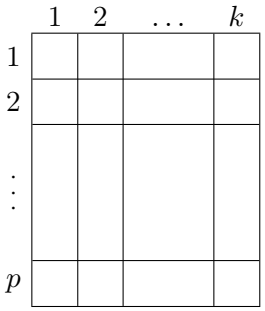

Рис. 2. Массив данных

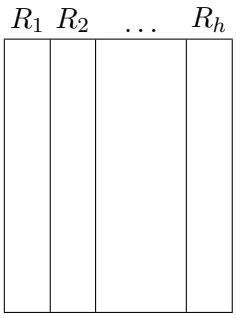

Рис. 3. Устройство ассоциативной обработки

Устройство ассоциативной обработки (Рис. 3) представляется в виде совокупности h вертикальных одноразрядных регистров длины p. Вертикальный регистр можно представлять как массив, состоящий из одного столбца. Обработка информации происходит следующим образом. Из определенной таблицы в определенном порядке извлекаются ее одноразрядные столбцы и помещаются в вертикальные регистры устройства ассоциативной обработки, в котором выполняются побитовые операции. Результат выполнения любой операции записывается либо в некоторый регистр, либо в определенный столбец обрабатываемой таблицы, либо в матричную память.

Чтобы моделировать обработку информации в матричной памяти, STAR-машина использует типы данных **slice, word** и **table**. С помощью переменной типа **slice** моделируется доступ к таблице по столбцам, а с помощью типа **word** – доступ по строкам. С каждой переменной типа **table** ассоциируется матрица из n строк и k столбцов, где $n \leq p$. С каждой переменной типа **slice** ассоциируется последовательность из p компонентов, принадлежащих множеству $\{0, 1\}$. Для простоты переменную типа **slice** условимся называть *слайсом*.

Пусть X и Y – слайсы, а i, j – переменные типа **integer**. Приведем некоторые операции и предикаты для слайсов.

SET(Y) записывает в слайс Y все единицы.

CLR(Y) записывает в слайс Y все нули.

$Y(i)$ выделяет i-й компонент в слайсе Y $(1 \leq i \leq p)$.

FND(Y) выдает порядковый номер i позиции первой (или старшей) единицы в слайсе Y, где $i \geq 0$.

STEP(Y) выдает такой же результат, как и операция FND(Y), и затем “гасит"(сбрасывает) старшую единицу.

CONVERT(Y) возвращает строку, каждый i-й бит которой совпадает с $Y(i)$. Эта операция применяется, когда строка одной матрицы используется как битовый столбец в другой матрице.

Операции FND(Y), STEP(Y) и CONVERT(Y) используются только в правой части оператора присваивания, в то время как операция $Y(i)$ может использоваться в любой части этого оператора.

Для выполнения параллелизма по данным общепринятым способом вводятся следующие побитовые логические операции: $X\,and\,Y$, $X\,or\,Y$, $not\,X$ и $X\,xor\,Y$. Так-

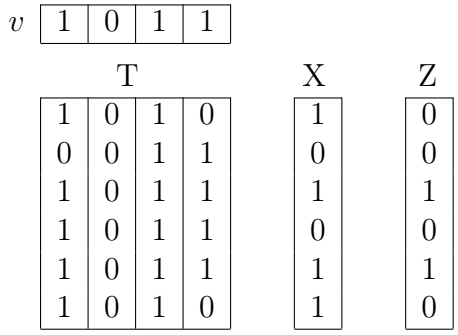

Рис. 4. Выполнение процедуры MATCH

же будем использовать предикат SOME(Y), который возвращает **true** тогда и только тогда, когда в слайсе Y имеется хотя бы один компонент $'1'$. Условимся, что запись $Y \neq \emptyset$ означает, что предикат SOME(Y) возвращает значение **true**.

Заметим, что предикат SOME(Y) и все операции для типа **slice** также используются для переменных типа **word**.

Пусть T – переменная типа **table**. Будем использовать следующие две операции:

ROW(i, T) выделяет i-ю строку в матрице T.

COL(i, T) выделяет i-й столбец в T.

Заметим, что операторы языка STAR вводятся таким же образом, как в языке Pascal.

Приведем группу базовых процедур [11, 12], которые будем использовать в данной работе. С помощью глобального слайса X будем указывать *позиции* анализируемых строк в соответствующей процедуре. В работах [11, 12] показано, что каждая базовая процедура выполняется за время $O(k)$, где k – это число битовых столбцов в соответствующей матрице.

Процедура MATCH(T, X, v, Z) одновременно определяет позиции строк в заданной матрице T, которые совпадают с заданной строкой v (Рис. 4). Она возвращает слайс Z, в котором $Z(i) =' 1'$ тогда и только тогда, когда ROW(i, T) $= v$ и $X(i) =' 1'$.

Процедура MIN(T, X, Z) одновременно определяет позиции строк в заданной матрице T, в которых записан минимальный элемент. Она возвращает слайс Z, в котором $Z(i) =' 1'$ тогда и только тогда, когда ROW(i, T) – минимальный элемент и $X(i) =' 1'$.

Процедура WCOPY(v, X, F) записывает двоичное слово v в те строки матрицы F, которые отмечены $'1'$ в слайсе X. Остальные строки матрицы F состоят из нулей.

Процедура CLEAR(n, T) записывает нули во все n столбцов матрицы T.

Процедура TCOPY1(T, j, h, F) записывает в результирующую матрицу F h столбцов из матрицы T, начиная с $(1 + (j - 1)h)$-го столбца, где $j \geq 1$.

Процедура TMERGE(T, X, F) записывает строки матрицы T, отмеченные $'1'$ в слайсе X, в соответствующие строки матрицы F. Остальные строки матрицы F не меняются.

Процедура SETMIN(T, F, X, Z) определяет позиции строк матрицы T, которые меньше соответствующих строк матрицы F. Она возвращает слайс Z, в котором $Z(i) =' 1'$ тогда и только тогда, когда ROW(j, T) $<$ ROW(j, F) и $X(j) =' 1'$.

Процедура ADDV(T, F, X, R) записывает в матрицу R результат одновременного

сложения соответствующих строк матриц T и F, позиции которых отмечены $'1'$ в слайсе X. Этот алгоритм использует таблицу 5.1 из [9].

Процедура ADDC(T, X, v, F) одновременно добавляет двоичное слово v к тем строкам матрицы T, которые отмечены $'1'$ в слайсе X, и записывает результат в соответствующие строки матрицы F. Остальные строки матрицы F состоят из нулей.

3. Основные понятия

Пусть $G = (V, E, wt)$ – это *ориентированный взвешенный граф*, в котором $V = \{1, 2, \ldots, n\}$ – множество вершин, $E \subseteq V \times V$ – множество ориентированных ребер (*дуг*) и $wt(e)$ – функция веса. Полагаем, что $|V| = n$ и $|E| = m$. Будем рассматривать графы, дуги которых имеют положительный вес. Будем считать, что $wt(i, j) = \infty$, если $(i, j) \notin E$.

Для рассматриваемой задачи значение бесконечности выбирается следующим образом: $\sum_{i=1}^{n} c_i$, где c_i – максимальный вес дуг, выходящих из вершины i в G. Обозначим через inf двоичный код величины этой суммы, а через h – число битов, используемых для кодирования этой величины.

Пусть $e = (i, j)$ – это дуга, которая ориентирована от вершины i до вершины j. При этом вершина j называется *головой* дуги e, а вершина i называется ее *хвостом*.

Матрицей смежности назовем квадратную матрицу $A = [a_{ij}]$ размера $n \times n$, элементы которой определяются следующим образом: $a_{ij} = 1$, если $(i, j) \in E$ и $a_{ij} = 0$, в противном случае.

Кратчайшим путем из вершины v_1 до вершины v_k в графе G назовем последовательность вершин v_1, v_2, …, v_k, в которой $(v_i, v_{i+1}) \in E$ для $1 \leq i < k - 1$ и сумма весов его дуг минимальна. Обозначим через $dist(l)$ *длину* кратчайшего пути из вершины v_1 до вершины l.

Деревом кратчайших путей T_s с корнем s назовем связный подграф без циклов, который содержит все вершины графа G, и путь от корня до любой вершины i является *минимальным*.

Пусть дуга (i, j) удалена из графа G. Вершина y называется *аффектной* после удаления дуги (i, j) из дерева кратчайших путей T_s, если вершина y не достижима из корня s.

4. Структура данных

В данном разделе вначале приведем структуру данных, которая будет использоваться в работе, а затем покажем, как можно ее получить.

Будем использовать следующую структуру данных:

- матрица смежности G размером $n \times n$, каждый i-й столбец которой хранит с помощью $'1'$ головы дуг, выходящих из вершины i;
- матрица SPT размером $n \times n$, каждый i-й столбец которой хранит с помощью $'1'$ головы дуг, выходящих из вершины i и принадлежащих дереву кратчайших путей;

- матрица $Weight$ размером $n \times hn$, элементами которой являются веса дуг. Она состоит из n полей, причем каждое поле состоит из h битов. Вес дуги (i, j) записывается в j-ю строку i-го поля;
- матрица $Cost$ размером $n \times hn$, элементами которой являются веса дуг. Она состоит из n полей, причем каждое поле состоит из h битов. Вес дуги (i, j) записывается в i-ю строку j-го поля;
- матрица $Dist$ размером $n \times h$, каждая i-я строка которой хранит кратчайшее расстояние от корня до вершины i;
- слайс $AffectedV$, который хранит с помощью $'1'$ позиции аффектных вершин.

Будем использовать следующие два свойства:

Свойство 1. Каждое i-е поле матрицы $Weight$ хранит веса дуг, выходящих из вершины i, а i-е поле матрицы $Cost$ хранит веса дуг, заходящих в эту вершину.

Свойство 2. В каждой i-й строке матриц G и SPT отмечены $'1'$ хвосты дуг, заходящих в вершину i.

Поясним, как построить приведенную выше структуру данных.

Исходный граф задаем в виде матрицы весов $Weight$ и матрицы смежности G, которая получается из матрицы весов с помощью вспомогательной процедуры Adj [13]. Матрица $Cost$ легко получается из матрицы $Weight$ с помощью простой вспомогательной процедуры, записанной на языке STAR. Слайс $AffectedV$ получаем с помощью вспомогательной процедуры AffectedVertSPT, которая будет приведена в данной статье. Матрицу кратчайших расстояний $Dist$ и матрицу SPT для дерева кратчайших путей будем получать с помощью вспомогательной процедуры DistSPT, которая является специальной реализацией на STAR–машине классического алгоритма Дейкстры [8] для нахождения кратчайших расстояний. В работе [13] приводится реализация алгоритма Дейкстры, позволяющая одновременно строить матрицу кратчайших расстояний $Dist$ и матрицу кратчайших путей $Paths$. В каждом i-м столбце матрицы $Paths$ хранятся номера вершин, которые входят в кратчайший путь от корня до вершины i.

Напомним, что алгоритм Дейкстры [8] присваивает временные метки $l(v)$ каждой вершине v графа G таким образом, что $l(v) \geq dist(s, v)$. Вначале $T_s = \{s\}$, $l(s) = 0$, и $\forall v \in T_s\ l(v) = \infty$. Эти метки постоянно уменьшаются с помощью итерационной процедуры и на каждой итерации только одна метка становится инвариантной. При этом соответствующая вершина включается в дерево кратчайших путей T_s. Процесс завершается, когда все вершины будут включены в дерево T_s.

Приведем процедуру DistSPT.

```
procedure DistSPT(Weight,Cost: table; s,h,n: integer;
  inf: word(Dist); var Dist: table; var SPT: table);
var A,R1,R2: table;
  U,X,Y,Z: slice(Weight);
  i,j,k: integer;
  v: word(Dist); v1: word(SPT);
1. Begin ADJ(Weight,h,n,inf,A);
2.  CLEAR(n,SPT);
```

```
 3.  SET(U); U(s):='0';
 4.  WCOPY(inf,U,Dist); k:=s;
 /* Переменная k является последней вершиной, включенной
    в дерево кратчайших путей T_s. */
 5.  while SOME(U) do
 6.    begin TCOPY1(Weight,k,h,R1);
 7.      X:=COL(k,A); X:=X and U;
 /* Слайс X хранит головы дуг, выходящих из вершины
    v_k и не принадлежащих дереву кратчайших путей. */
 8.      v:=ROW(k,Dist);
 9.      ADDC(R1,X,v,R2);
 /* Значение l(v_k)+wt(v_k,v_i) записывается в те строки
    матрицы R2, которые отмечены '1' в слайсе X. */
10.      SETMIN(R2,Dist,X,Z);
11.      TMERGE(R2,Z,Dist);
 /* В каждой i-й строке матрицы Dist, отмеченной '1'
    в слайсе Z, значение l(v_i) заменяется на l(v_k)+wt(v_k,v_i). */
12.      MIN(Dist,U,Y); k:=FND(Y);
13.      U(k):='0';
 /* Новая вершина включается в дерево T_s. */
14.      v1:=ROW(k,A); X:=CONVERT(v1);
15.      X:=X and (not U);
 /* В слайсе X отмечены '1' хвосты дуг, заходящих в вершину
    v_k и принадлежащих дереву T_s. */
16.      TCOPY1(Cost,k,h,R1);
 /* В матрицу R1 записывается k-е поле матрицы Cost. */
17.      ADDV(R1,Dist,X,R2);
 /* Веса путей от корня до вершины v_k записаны в строках
    матрицы R2, отмеченных '1' в слайсе X. */
18.      MIN(R2,X,Z); i:=FND(Z);
 /* Кратчайший путь от корня до v_k включает вершину v_i,
    которая является хвостом дуги, заходящей в вершину v_k. */
19.      X:=COL(i,SPT); X(k):='1';
20.      COL(i,SPT):=X;
21.    end;
22. End;
```

Теорема 1. *Пусть заданы матрицы $Weight$ и $Cost$, целые числа s, h, n и двоичное слово inf, которое хранит код бесконечности. Тогда процедура* DistSPT *возвращает матрицу $Dist$ и матрицу SPT.*

Доказательство. Поскольку процедура DistSPT определяет матрицу $Dist$ таким же образом, как процедура DistPaths [13], то достаточно проверить, что процедура DistSPT строит матрицу SPT. Будем доказывать индукцией по числу дуг q, включенных в дерево T_S.

Базис индукции проверяем для $q = 1$. После выполнения строк 1–4 построена матрица смежности A для матрицы $Weight$, матрица SPT состоит из нулей,

вершина s включена в дерево T_s, поскольку $U(s) =' 0'$, в s-й строке матрицы $Dist$ записано расстояние от s до s, и оно равно нулю, а в остальных строках этой матрицы записано слово inf и переменная k хранит вершину s. Поскольку $U \neq \emptyset$, то выполняем цикл `while SOME(U) do` из строки 5. После выполнения строк 6–13 слайс X хранит головы дуг, выходящих из вершины s, в каждой i-й строке матрицы $Dist$, отмеченной $'1'$ в слайсе Z, значение $l(v_i)$ заменяется на меньшее значение $l(s) + wt(s, v_i)$, а в дерево T_s включается та вершина v_k, для которой данная сумма имеет минимальное значение. После выполнения строк 14–15 запоминаем в слайсе X те вершины из дерева T_s, которые являются хвостами дуг, заходящими в новую вершину v_k. Так как в T_s, кроме новой вершины, v_k имеется вершина s, то она будет хвостом дуги (s, v_k). После выполнения строк 16–20 эта дуга добавится в матрицу SPT путем записи единицы в k-й бит s-то столбца.

Шаг индукции. Пусть утверждение выполняется для $q \geq 1$. Докажем его для случая, когда $q+1$ дуг будут включены в дерево T_s. По индуктивному предположению после включения первых q дуг в дерево T_s их позиции будут отмечены нулем в слайсе U, переменная k будет хранить текущую вершину, добавленную в T_s, и вместе с каждой вершиной v_j, включенной в T_s, определяется дуга (v_i, v_j), которая добавляется в дерево T_s и в матрицу SPT. Поскольку $U \neq \emptyset$, то выполняем цикл `while SOME(U) do` из строки 5 для $(q+1)$-й вершины. Далее рассуждаем аналогично базису индукции. После включения в дерево T_s и в матрицу SPT дуги минимального веса, заходящей в $(q+1)$-ю вершину, слайс U будет состоять из нулей. Поэтому попадаем на конец этой процедуры.

Теорема доказана.

5. Декрементальный ассоциативный алгоритм для обработки дерева кратчайших путей

В данном разделе вначале приведем содержательное описание ассоциативного декрементального алгоритма для обработки дерева кратчайших путей ориентированного графа. Пусть дуга (i, j) удаляется из T_s. Поскольку в каждую вершину дерева входит единственная дуга, то вершина j становится аффектной или недостижимой из корня s. А тогда аффектными вершинами будут те вершины дерева T_s, которые принадлежат поддереву с корнем j. Вначале декрементальный алгоритм определяет множество всех аффектных вершин, которые образуются после удаления дуги (i, j) из T_s. Затем определяется новое расстояние от корня до каждой аффектной вершины. После этого будут перевычисляться расстояния до тех аффектных вершин k, которые являются головами дуг, выходящих из вершины k, и для которых новые расстояния от корня уменьшаются. Для выполнения этих этапов рассматриваемой задачи будут приведены три ассоциативных параллельных алгоритма. Отметим, что декрементальный ассоциативный алгоритм для динамической обработки дерева кратчайших путей использует идею, приведенную в декрементальном алгоритме Рамалингама [1].

Приведем ассоциативный параллельный алгоритм для нахождения аффектных вершин и аффектных дуг (скажем, *Алгоритм 1*), которые получаются после удаления дуги (i, j) из T_s. Алгоритм будет находить вершины, которые принадлежат

компоненте связности, включающей вершину j. Он будет использовать матрицу SPT, слайс $AffectedV$ и вспомогательный слайс Z. Ассоциативный параллельный алгоритм выполняет следующие шаги:

Шаг 1. Включить в слайсы $AffectedV$ и Z головы дуг, выходящих из вершины j. Включить вершину j в слайс $AffectedV$. Удалить из матрицы SPT все дуги, выходящие из вершины j.

Шаг 2. Пока $Z \neq \emptyset$, выполнить следующие действия:

– удалить позицию самого верхнего бита $'1'$ (скажем r) из слайса Z;

– включить в слайсы $AffectedV$ и Z головы дуг, выходящих из вершины r;

– удалить все дуги, выходящие из вершины r в матрице SPT.

На STAR–машине этот алгоритм представляется в виде процедуры AffectedVertSPT, которая возвращает слайс $AffectedV$ и измененную матрицу SPT.

Ассоциативный параллельный алгоритм для вычисления нового расстояния от корня до каждой аффектной вершины (скажем, *Алгоритм 2*) выполняется следующим образом.

Пока $AffectedV \neq \emptyset$, определять новое расстояние от корня до каждой аффектной вершины с помощью следующих шагов.

Шаг 1. Выделить позицию текущей обрабатываемой вершины k в слайсе $AffectedV$ и пометить ее нулем.

Шаг 2. Вычислить в графе G веса различных путей от корня до вершины k, которые не содержат аффектных вершин.

Шаг 3. Определить минимальное расстояние от корня до вершины k и записать его в k-ю строку матрицы $Dist$.

На STAR–машине этот алгоритм реализован в виде процедуры NewDistSPT, которая возвращает измененную матрицу $Dist$.

Ассоциативный параллельный алгоритм для обработки дуг, выходящих из аффектной вершины k, (скажем, *Алгоритм 3*) выполняет следующие шаги:

Шаг 1. Зная слайс $AffectedV$ и кратчайшее расстояние до вершины k, определить в матрице G вес пути от корня до каждой аффектной вершины r, которая является головой дуги, выходящей из вершины k.

Шаг 2. С помощью слайса (скажем, Y) сохранить головы дуг, выходящих из вершины k, для которых $dist_{new}(r) < dist_{old}(r)$. Записать новые расстояния в соответствующие строки матрицы $Dist$.

На STAR–машине этот алгоритм реализован в виде процедуры LeavingArcsSPT.

Теперь приведем ассоциативный параллельный алгоритм для динамической обработки дерева кратчайших путей после удаления дуги (i, j) из графа G. Он выполняет следующие шаги.

Шаг 1. Удалить из матрицы G позицию дуги (i, j). Если $(i, j) \notin SPT$, то перейти на выход. В противном случае удалить позицию дуги (i, j) из матрицы SPT.

Шаг 2. С помощью *Алгоритма 1* построить слайс $AffectedV$ и удалить аффектные дуги из матрицы SPT.

Шаг 3. С помощью *Алгоритма 2* определить новые расстояния от корня до всех аффектных вершин и записать их в соответствующие строки матрицы $Dist$.

Шаг 4. Пока $AffectedV \neq \emptyset$, обработать все аффектные вершины следующим образом:

– определить аффектную вершину k, которая удалена от корня на минимальное расстояние, и удалить вершину k из слайса $AffectedV$;

– определить хвост r дуги, заходящей в вершину k, и включить позицию дуги (r, k) в матрицу SPT;

– с помощью *Алгоритма 3* перевычислить расстояния от корня до тех аффектных вершин q, которые являются головами дуг, выходящих из вершины k в графе G и $dist_{new}(q) < dist_{old}(q)$. Записать $dist_{new}(q)$ в соответствующие строки матрицы $Dist$.

На STAR–машине этот алгоритм реализован в виде процедуры DeleteArcSPT.

6. Выполнение декрементального ассоциативного алгоритма на STAR–машине

В данном разделе вначале рассмотрим выполнение вспомогательных процедур на STAR-машине, а затем приведем основную процедуру DeleteArcSPT.

Приведем вспомогательную процедуру AffectedVertSPT. Зная вершины i и j и текущую матрицу SPT, она возвращает слайс $AffectedV$ и удаляет из матрицы SPT аффектные дуги.

```
procedure AffectedVertSPT(i,j: integer; var SPT: table;
  var AffectedV: slice(G));
/* Дуга (i,j) удалена из матриц G и SPT. */
var X,Y,Z: slice(G);
  r: integer;
1. Begin CLR(Y); AffectedV:=COL(j,SPT);
2.  Z:=COL(j,SPT); AffectedV(j):='1';
3.  COL(j,SPT):=Y;
/* Записываем нули в j-й столбец матрицы SPT. */
4.  while SOME(Z) do
5.    begin r:=STEP(Z);
6.      X:=COL(r,SPT); COL(r,SPT):=Y;
/* Удаляем из SPT дуги, выходящие из вершины r. */
7.      AffectedV:=AffectedV or X;
8.      Z:=Z or X;
9.    end;
10. End.
```

Лемма 1. *Пусть дуга (i, j) удалена из дерева кратчайших путей T_s. Тогда процедура* AffectedVertSPT *возвращает слайс AffectedV, в котором позиции аффектных вершин отмечены '1'. Кроме того, все дуги, выходящие из каждой аффектной вершины, будут удалены из матрицы SPT.*

Доказательство. Надо доказать, что после выполнения процедуры AffectedVertSPT каждая вершина l из поддерева T_j принадлежит слайсу $AffectedV$ и l-й столбец матрицы SPT состоит из нулей. Будем доказывать индукцией по высоте q поддерева T_j.

Базис индукции проверяется для $q = 1$. После выполнения строк 1–3 слайс Y состоит из нулей, головы дуг, выходящих из вершины j, отмечены $'1'$ в слайсе Z, вершина j и головы дуг, выходящих из нее, отмечены $'1'$ в слайсе $AffectedV$ и j-й столбец матрицы SPT состоит из нулей. Теперь выполняем цикл `while SOME(Z) do` (строки 5–9). После выполнения строк 5–6 слайс $X = \emptyset$, поскольку каждая вершина r, отмеченная $'1'$ в слайсе Z, является листом в T_j и r-й столбец матрицы SPT состоит из нулей. После выполнения строк 7–8 никакая новая вершина не включается в слайсы $AffectedV$ и Z. После обработки последней вершины, отмеченной $'1'$ в слайсе Z, получаем, что $Z = \emptyset$. Поэтому переходим на конец процедуры.

Шаг индукции. Пусть утверждение справедливо для любого поддерева T_l высоты $q \geq 1$. Докажем лемму для поддерева T_j высоты $q+1$. По индуктивному предположению после обработки любого поддерева T_l высоты q, все вершины, принадлежащие поддереву T_l, будут включены в слайс $AffectedV$ и все дуги, выходящие из каждой вершины T_l, будут удалены из матрицы SPT.

Рассмотрим поддерево дерева T_j высоты $q+1$. Как и при доказательстве базиса индукции после выполнения строк 1–3, слайс $AffectedV$ будет хранить вершину j и головы дуг, которые выходят из j, а все дуги, выходящие из вершины j, будут удалены из матрицы SPT. Заметим, что голова r любой дуги (j, r) в поддереве T_j становится корнем поддерева T_r высоты не более, чем q. Поэтому применяем индуктивное предположение к каждому такому поддереву. После обработки последней вершины, отмеченной $'1'$ в слайсе Z, получаем, что слайс $AffectedV$ будет хранить все вершины поддерева T_j, а все дуги, выходящие из всех вершин этого поддерева, будут удалены из матрицы SPT.

Лемма 1 доказана.

Приведем процедуру NewDistSPT, которая определяет новые расстояния от корня s до остальных аффектных вершин.

```
procedure NewDistSPT(h: integer; G: table; Cost: table;
  AffectedV: slice(G); var Dist: table);
var v: word(G); v1: word(Dist);
  X,Z,Z1: slice(G);
  W1,W2: table;
  k,r: integer;
1. Begin X:=AffectedV;
2.  while SOME(X) do
3.    begin k:=FND(X); v:=ROW(k,G);
4.      Z:=CONVERT(v);
/* Слайс Z хранит хвосты дуг, заходящих в вершину k в графе G. */
5.      Z1:=Z and (not AffectedV);
/* Слайс Z1 хранит те вершины из Z, которые
  не являются аффектными. */
6.      TCOPY1(Cost,k,h,W1);
7.      ADDV(Dist,W1,Z1,W2);
/* Матрица W2 хранит различные расстояния от s до k. */
8.      MIN(W2,Z1,Z);
9.      r:=FND(Z); v1:=ROW(r,W2);
10.     ROW(k,Dist):=v1;
```

```
  /* Новое расстояние от s до k записывается в k-ю строку
    матрицы Dist. */
11.       X(k):='0';
12.     end;
13. End.
```

Лемма 2. *Пусть заданы число h, слайс $AffectedV$ и текущие матрицы G, $Cost$ и $Dist$. Тогда процедура* NewDistSPT *возвращает измененную матрицу $Dist$, которая хранит новые кратчайшие расстояния от корня до каждой аффектной вершины.*

Доказательство. Будем доказывать индукцией по числу аффектных вершин l.

Базис индукции проверяем для $l = 1$. После выполнения строк 1–4 слайс X хранит единственную аффектную вершину, $k = j$ и слайс Z хранит хвосты дуг, заходящих в вершину k в графе G. После выполнения строк 5–7 матрица $W2$ хранит веса различных путей, ведущих из корня до вершины k, причем каждый такой путь не содержит аффектных вершин. После выполнения строк 8–10 вначале определяем хвост дуги (r, k), принадлежащей новому кратчайшему пути от корня до вершины k, а затем записываем это кратчайшее расстояние в k-ю строку матрицы $Dist$. После выполнения строки 11 переходим на выход, поскольку $X = \emptyset$.

Шаг индукции. Пусть утверждение справедливо для l ($l \geq 1$) аффектных вершин. Докажем его для $l + 1$ таких вершин. По индуктивному предположению после обработки первых l аффектных вершин новое расстояние от корня до каждой такой вершины будет записано в соответствующей строке матрицы $Dist$, а в слайсе X останется необработанной только одна аффектная вершина. Далее рассуждаем аналогично базису индукции.

Лемма 2 доказана.

Приведем процедуру LeavingArcsSPT, которая использует текущую обрабатываемую вершину k, слайс $AffectedV$ и матрицы $Weight$ и $Dist$.

```
procedure LeavingArcsSPT(h,k: integer; G: table; Weight: table;
  var Dist: table);
var v: word(Dist);
  W1,W2: table;
  Y,Z: slice(Dist);
1. Begin Z:=COL(k,G);
2.  TCOPY1(Weight,k,h,W1);
/* Матрица W1 хранит веса дуг, выходящих из вершины k. */
3.  v:=ROW(k,Dist);
4.  ADDC(W1,Z,v,W2);
/* Матрица W2 хранит новые расстояния от корня до тех
  вершин p, которые принадлежат дуге (k,p). */
5.  SETMIN(W2,Dist,Z,Y);
6.  TMERGE(W2,Y,Dist);
7. End;
```

Лемма 3. *Пусть заданы число h, номер текущей обрабатываемой вершины k, слайс $AffectedV$ и текущие матрицы $Weight$ и $Dist$. Тогда процедура* LeavingArcsSPT

возвращает измененную матрицу Dist, где новые кратчайшие расстояния записываются для тех вершин r, которые являются головами дуг с общим хвостом k, и для которых значения $dist_{new}(r)$ уменьшаются.

Доказательство. Будем доказывать методом от противного. Пусть дуга (k, r) принадлежит графу G и $dist_{new}(r) < dist_{old}(r)$. Однако после выполнения процедуры LeavingArcsSPT r-я строка матрицы $Dist$ не изменилась. Докажем, что это противоречит выполнению нашей процедуры.

Действительно, поскольку $(k, r) \in G$, то получаем, что $Z(r) =' 1'$ после выполнения строки 1. После выполнения строк 2–4 вес кратчайшего пути от корня до вершины r, включающего дугу (k, r), будет записан в r-ю строку матрицы $W2$. По предположению $dist_{new}(r) < dist_{old}(r)$. Поэтому после выполнения базовой процедуры SETMIN из строки 5 получаем $Y(r) =' 1'$. А тогда после выполнения строки 6 новое расстояние $dist_{new}(r)$ до вершины r будет записано в r-й строке матрицы $Dist$. Это противоречит нашему допущению.

Теперь приведем основную процедуру DeleteArcSPT, которая возвращает обработанные матрицы G, SPT и $Dist$.

```
procedure DeleteArcSPT(i,j,h: integer; Weight,Cost: table;
  var G,SPT: table; var Dist: table);
/* Дуга (i,j) будет удаляться из графа G. */
var k,r: integer;
  AffectedV,X,Y,Z: slice(G);
  W1,W2: table;
  v: word(G); v1: word(Dist);
  label 1;
1. Begin X:=COL(i,G); X(j):='0';
2.  COL(i,G):=X;
/* Дуга (i,j) удаляется из G. */
3.  X:=COL(i,SPT);
4.  if X(j)='0' then goto 1;
5.  X(j):='0'; COL(i,SPT):=X;
/* Дуга (i,j) удаляется из SPT. */
6.  AffectedVertSPT(i,j,SPT,AffectedV);
/* Эта процедура возвращает матрицу SPT и слайс AffectedV. */
7.  NewDistSPT(h,G,Cost,AffectedV,Dist);
/* Эта процедура возвращает измененную матрицу Dist. */
8.  while SOME(AffectedV) do
9.    begin MIN(Dist,AffectedV,Z);
/* Слайс Z хранит позиции тех строк матрицы Dist,
  в которых записано минимальное значение. */
10.       k:=FND(Z); AffectedV(k):='0';
/* Вершина k включается в дерево кратчайших путей. */
11.       v:=ROW(k,G); Z:=CONVERT(v);
12.       X:=Z and (not AffectedV);
/* Слайс X хранит хвосты тех дуг, которые заходят в вершину
  k в графе G и не являются аффектными. */
```

```
13.        v1:=ROW(k,Dist);
14.        TCOPY1(Cost,k,h,W1);
15.        ADDV(Dist,W1,X,W2);
 /* Матрица W2 хранит различные расстояния от s до k в G. */
16.        MATCH(W2,X,v1,Y); r:=FND(Y);
17.        Y:=COL(r,SPT); Y(k):='1';
18.        COL(r,SPT):=Y;
 /* Дуга (r,k) включается в матрицу SPT. */
19.        LeavingArcsSPT(h,k,Weight,AffectedV,Dist);
 /* Эта процедура записывает новые расстояния в матрицу Dist для
   тех вершин q, для которых уменьшается dist_new(q) и (k,q) ∈ E. */
20.    end;
21. 1: End;
```

Теорема 2. *Пусть ориентированный граф задается в виде матрицы смежности G и матрицы весов Weight. Пусть также заданы матрицы Cost, SPT и Dist и числа n и h. Пусть из графа удалена дуга (i,j). Тогда процедура* DeleteArcSPT *возвращает обработанные матрицы G, SPT и Dist.*

Доказательство. Будем доказывать индукцией по числу аффектных вершин q, которые появляются после удаления дуги (i, j) из дерева кратчайших путей.

Базис индукции доказывается для $q = 1$. Непосредственно проверяется, что после выполнения строк 1–5 позиция дуги (i, j) удаляется из матриц G и SPT. После выполнения строки 6 ввиду леммы 1 слайс $AffectedV$ хранит позицию аффектной вершины j и все дуги, выходящие из вершины j, удалены из матрицы SPT. Заметим, что j-й столбец матрицы SPT будет состоять из нулей, поскольку j – единственная аффектная вершина в T_s. После выполнения строки 7 ввиду леммы 2 получаем, что $AffectedV(j) =' 1'$ и новое кратчайшее расстояние от корня до вершины j будет записано в j-й строке матрицы $Dist$. Поскольку $AffectedV \neq \theta$, то выполняем цикл `while SOME(AffectedV) do` (строка 8). После выполнения строк 9–10 получаем, что $k = j$, $AffectedV = \theta$ и вершина j включена в дерево кратчайших путей. Чтобы найти хвост дуги, заходящей в вершину j, которую надо включить в матрицу SPT, будем использовать метод, предложенный в статье [13]. Вначале находим хвосты дуг, заходящих в вершину j в матрице G, которые не являются аффектными (строки 11–12). После выполнения строк 13–15 определяем различные расстояния от корня до вершины j, которые включают выделенные дуги. После выполнения строки 16 определяем вершину r, на которой достигается минимальное расстояние от корня. Наконец после выполнения строк 17–18 новая дуга (r, j) включается в матрицу SPT.

Шаг индукции. Пусть утверждение выполняется, когда q ($q \geq 1$) аффектных вершин обработаны в графе G. Докажем утверждение для $q+1$ аффектных вершин.

Непосредственно проверяется, что после выполнения строк 1–7 дуга (i, j) удаляется из матриц G и SPT, слайс $AffectedV$ хранит позиции $q + 1$ аффектных вершин, все аффектные дуги удалены из матрицы SPT и новые расстояния от корня до всех аффектных вершин записаны в соответствующие строки матрицы $Dist$. Поскольку $AffectedV \neq \theta$, выполняем строку 8.

После выполнения строк 9–10 определяем позицию аффектной вершины k, для которой расстояние от корня имеет минимальное значение, и отмечаем ее позицию нулем в слайсе $AffectedV$. Аналогично базису после выполнения строк 11–18 вначале определяем хвост новой дуги (r, k), которая будет принадлежать новому кратчайшему пути от корня до вершины k. Затем включаем эту дугу в матрицу SPT. Ввиду леммы 3 после выполнения строки 19 в матрицу $Dist$ записываем новые расстояния от корня до тех вершин r, которые принадлежат дугам, заходящим в вершину k, и для которых уменьшаются значения $dist_{new}(r)$.

Теперь мы имеем только q аффектных вершин, позиции которых отмечены $'1'$ в слайсе $AffectedV$. По индуктивному предположению после обработки q аффектных вершин новое кратчайшее расстояние от корня до каждой аффектной вершины r записывается в r-ю строку матрицы $Dist$ и новая дуга, заходящая в вершину r, записывается в матрицу SPT. Кроме того, после включения каждой очередной вершины p в T_s мы перевычисляем в матрице $Dist$ веса путей от корня до тех вершин t, которые принадлежат дугам, выходящим из вершины p, чьи новые расстояния от корня уменьшаются. Это гарантирует нам правильный выбор нового кратчайшего пути от корня до каждой аффектной вершины, которая будет включаться в T_s.

Теорема доказана.

Оценим время выполнения процедуры DeleteArcSPT. Вначале оценим время выполнения вспомогательных процедур. Пусть h – число битов для кодирования бесконечности, а k – число аффектных вершин, которые появляются после удаления дуги (i, j) из дерева кратчайших путей. Вспомогательная процедура AffectedVertSPT выполняется за время $O(k)$. Вспомогательная процедура NewDistSPT выполняется за время $O(kh)$, поскольку она использует базовые процедуры, каждая из которых выполняется за время $O(h)$. Вспомогательная процедура LeavingArcsSPT выполняется за время $O(h)$. В процедуре DeleteArcSPT основной цикл `while SOME(AffectedV) do` (строки 8–20) выполняется за время $O(kh)$, поскольку внутри этого цикла каждая используемая базовая процедура и вспомогательная процедура LeavingArcsSPT выполняются за время $O(h)$.

7. Заключение

В статье построен ассоциативный параллельный алгоритм для динамической обработки дерева кратчайших путей после удаления из ориентированного графа одной дуги. Этот алгоритм использует простую структуру данных, которая позволяет выполнять доступ к данным по содержимому памяти. Ассоциативный алгоритм представлен на STAR–машине в виде процедуры DeleteArcSPT, корректность которой доказана. Мы показали, что эта процедура выполняется на STAR–машине за время $O(hk)$, где h – число битов для кодирования бесконечности, а k – число аффектных вершин, для которых строятся новые кратчайшие пути. Полученная оценка является оптимальной, поскольку параметр h не меняется с ростом k, а обрабатывать надо все аффектные вершины. Реализация этой процедуры упрощена, в частности, за счет того, что аффектные вершины образуют компоненту связности, включающую голову удаленной дуги.

Планируется построить ассоциативный параллельный алгоритм для динамической обработки дерева кратчайших путей после добавления новой дуги в ориентированный граф.

Список литературы

1. Ramalingam G. Bounded Incremental Computation // Lecture Notes in Computer Science. Berlin: Springer–Verlag, 1996. Vol. 1089. P. 30–51.

2. Ramalingam G., Reps T. An incremental algorithm for a generalization of the shortest paths problem // Journal of Algorithms. Academic Press. 1996. Vol. 21. P. 267–305.

3. Frigioni D., Marchetti-Spaccamela A., Nanni U. Semi–dynamic algorithms for maintaining single source shortest paths trees // Algorithmica. Berlin: Springer–Verlag, 1998. Vol. 25. P. 250–274.

4. Frigioni D., Marchetti-Spaccamela A., Nanni U. Fully dynamic algorithms for maintaining shortest paths trees // Journal of Algorithms. Academic Press. 2000. Vol. 34. P. 351–381.

5. Frigioni D., Marchetti-Spaccamela A., Nanni U. Fully dynamic shortest paths in digraphs with arbitrary arc weights // Journal of Algorithms. Elsevier Science. 2003. Vol. 49. P. 86–113.

6. Nepomniaschaya A. S. Associative version of the Ramalingam decremental algorithm for dynamic updating the single-sink shortest paths subgraph // Proc. of the 10-th International Conference on Parallel Computing Technologies, PaCT-2009, Novosibirsk, Russia. Lecture Notes in Computer Science. Berlin: Springer-Verlag, 2009. Vol. 5698. P. 257–268.

7. Nepomniaschaya A. S. Language STAR for associative and parallel computation with vertical data processing // Proc. of the International Conference “Parallel Computing Technologies”. World Scientific, Singapore, 1991. P. 258–265.

8. Dijkstra E. W. A note on two problems in connection with graphs // Numerische Mathematik. 1959. Vol. 1. P. 269–271.

9. Foster C.C. Content Addressable Parallel Processors. New York: Van Nostrand Reinhold Company, 1976.

10. Непомнящая А. Ш., Владыко М. А. Сравнение моделей ассоциативных параллельных вычислениий // Программирование. 1997. № 6. С. 41–50 (English transl.: Nepomniaschaya A.Sh., Vladyko M.A. A Comparison of Associative Computation Models //Programming and Computer Software. 1997. V. 23, № 6. P. 319–324).

11. Nepomniaschaya A. S. Solution of path problems using associative parallel processors // Proc. of the Intern. Conf. on Parallel and Distributed Systems, ICPADS’97. Korea, Seoul, IEEE Computer Society Press, 1997. P. 610–617.

12. Nepomniaschaya A. S., Dvoskina M. A. A simple implementation of Dijkstra’s shortest path algorithm on associative parallel processors // Fundamenta Informaticae. IOS Press, 2000. Vol. 43. P. 227-243.

13. Nepomniaschaya A. S. Simultaneous finding the shortest paths and distances in directed graphs using associative parallel processors // Proc. of the Intern. Conf. "Information Visualization (IV 2003). England, London, IEEE Computer Society, Los Alamitos, 2003. P. 665–670.

Associative Parallel Algorithm for Dynamic Update of the Shortest Paths Tree

Nepomniaschaya A. S.

Institute of Computational Mathematics and Mathematical Geophysics SB RAS
prospect Akademika Lavrentjeva, 6, Novosibirsk, 630090, Russia

Keywords: directed weighted graph, the shortest paths tree, adjacency matrix, decremental algorithm, associative parallel processor

The paper proposes an efficient associative algorithm for dynamic update of the shortest paths tree of a directed weighted graph after deleting an edge. To this end, we provide the data structure and its building along with the STAR–machine that simulates the run of associative (content–addressable) parallel systems with simple single–bit processing elements and vertical processing. The associative algorithm is represented on the STAR–machine as the main procedure DeleteArcSPT that uses some auxiliary procedures. By means of the auxiliary procedures, we execute some parts of the associative parallel algorithm for dynamic update of the shortest paths tree after deleting an arc from the graph. We prove correctness of the procedure DeleteArcSPT and all its parts. We obtain that on the STAR–machine this procedure takes $O(hk)$ time, where h is the number of bits required for coding the maximum of the shortest paths weights and k is the number of vertices, whose shortest paths change after deleting an edge from the given graph.

Сведения об авторе:
Непомнящая Анна Шмилевна,
Институт вычислительной математики и математической геофизики СО РАН,
канд. физ.-мат. наук, старший научный сотрудник